Load The Dataset (Week 2)

In [1]:

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

#ingest data
df = pd.read_csv('https://raw.githubusercontent.com/Christine971224/Analytics-2023/mast
df.head()
```

Out[1]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	

5 rows × 36 columns

In [2]:

```
#basic information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 36 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           119390 non-null  object 
 1   is_canceled                     119390 non-null  int64  
 2   lead_time                       119390 non-null  int64  
 3   arrival_date_year               119390 non-null  int64  
 4   arrival_date_month              119390 non-null  object 
 5   arrival_date_week_number        119390 non-null  int64  
 6   arrival_date_day_of_month       119390 non-null  int64  
 7   stays_in_weekend_nights         119390 non-null  int64  
 8   stays_in_week_nights            119390 non-null  int64  
 9   adults                          119390 non-null  int64  
 10  children                        119386 non-null  float64
```

```
 11  babies                          119390 non-null  int64  
 12  meal                            119390 non-null  object 
 13  country                         118902 non-null  object 
 14  market_segment                  119390 non-null  object 
 15  distribution_channel            119390 non-null  object 
 16  is_repeated_guest               119390 non-null  int64  
 17  previous_cancellations          119390 non-null  int64  
 18  previous_bookings_not_canceled  119390 non-null  int64  
 19  reserved_room_type              119390 non-null  object 
 20  assigned_room_type              119390 non-null  object 
 21  booking_changes                 119390 non-null  int64  
 22  deposit_type                    119390 non-null  object 
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64  
 26  customer_type                   119390 non-null  object 
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64  
 29  total_of_special_requests       119390 non-null  int64  
 30  reservation_status              119390 non-null  object 
 31  reservation_status_date         119390 non-null  object 
 32  name                            119390 non-null  object 
 33  email                           119390 non-null  object 
 34  phone-number                    119390 non-null  object 
 35  credit_card                     119390 non-null  object 
dtypes: float64(4), int64(16), object(16)
memory usage: 32.8+ MB
```

In [3]:

```
df.isnull().mean()
```

Out[3]:

```
hotel                             0.000000
is_canceled                       0.000000
lead_time                         0.000000
arrival_date_year                 0.000000
arrival_date_month                0.000000
arrival_date_week_number          0.000000
arrival_date_day_of_month         0.000000
stays_in_weekend_nights           0.000000
stays_in_week_nights              0.000000
adults                            0.000000
children                          0.000034
babies                            0.000000
meal                              0.000000
country                           0.004087
market_segment                    0.000000
distribution_channel              0.000000
is_repeated_guest                 0.000000
previous_cancellations            0.000000
previous_bookings_not_canceled    0.000000
reserved_room_type                0.000000
assigned_room_type                0.000000
booking_changes                   0.000000
deposit_type                      0.000000
agent                             0.136862
company                           0.943069
days_in_waiting_list              0.000000
customer_type                     0.000000
adr                               0.000000
required_car_parking_spaces       0.000000
```

```
total_of_special_requests        0.000000
reservation_status               0.000000
reservation_status_date          0.000000
name                             0.000000
email                            0.000000
phone-number                     0.000000
credit_card                      0.000000
dtype: float64
```

In [4]:

```
# adults, babies and children can't be zero at same time, so dropping the rows having a
filter = (df.children == 0) & (df.adults == 0) & (df.babies == 0)
df[filter]
```

Out[4]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number
2224	Resort Hotel	0	1	2015	October	41
2409	Resort Hotel	0	0	2015	October	42
3181	Resort Hotel	0	36	2015	November	47
3684	Resort Hotel	0	165	2015	December	53
3708	Resort Hotel	0	165	2015	December	53
...	...	...	...	...	...	...
115029	City Hotel	0	107	2017	June	26
115091	City Hotel	0	1	2017	June	26
116251	City Hotel	0	44	2017	July	28
116534	City Hotel	0	2	2017	July	28
117087	City Hotel	0	170	2017	July	30

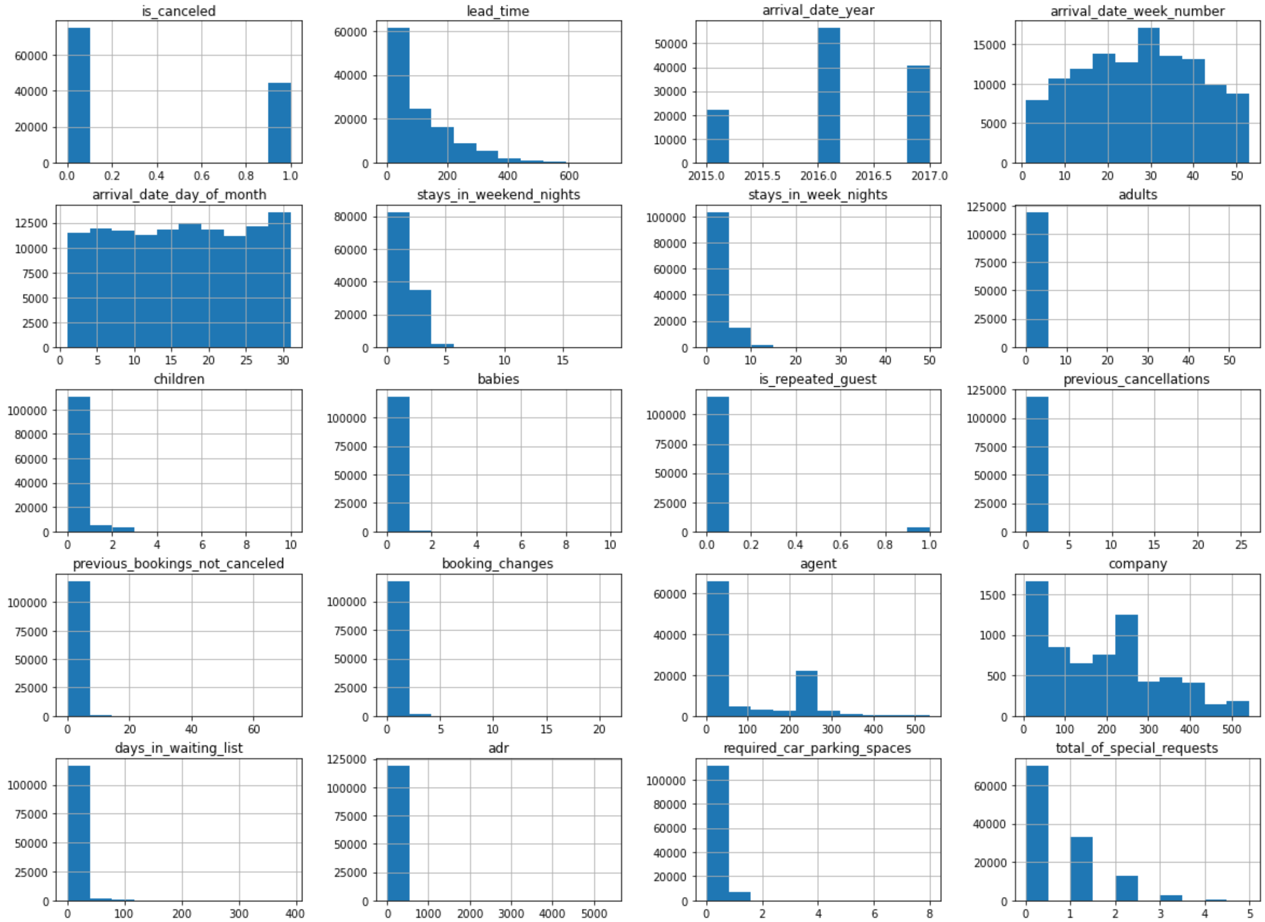
180 rows × 36 columns

In [5]:
```
# transpose the resulting DataFrame
df.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.99]).T
```

Out[5]:

	count	mean	std	min	1%	5%	10%	2!
is_canceled	119390.0	0.370416	0.482918	0.00	0.0	0.0	0.0	0
lead_time	119390.0	104.011416	106.863097	0.00	0.0	0.0	3.0	18
arrival_date_year	119390.0	2016.156554	0.707476	2015.00	2015.0	2015.0	2015.0	2016
arrival_date_week_number	119390.0	27.165173	13.605138	1.00	2.0	5.0	8.0	16
arrival_date_day_of_month	119390.0	15.798241	8.780829	1.00	1.0	2.0	4.0	8
stays_in_weekend_nights	119390.0	0.927599	0.998613	0.00	0.0	0.0	0.0	0
stays_in_week_nights	119390.0	2.500302	1.908286	0.00	0.0	0.0	1.0	1
adults	119390.0	1.856403	0.579261	0.00	1.0	1.0	1.0	2
children	119386.0	0.103890	0.398561	0.00	0.0	0.0	0.0	0
babies	119390.0	0.007949	0.097436	0.00	0.0	0.0	0.0	0
is_repeated_guest	119390.0	0.031912	0.175767	0.00	0.0	0.0	0.0	0
previous_cancellations	119390.0	0.087118	0.844336	0.00	0.0	0.0	0.0	0
previous_bookings_not_canceled	119390.0	0.137097	1.497437	0.00	0.0	0.0	0.0	0
booking_changes	119390.0	0.221124	0.652306	0.00	0.0	0.0	0.0	0
agent	103050.0	86.693382	110.774548	1.00	1.0	1.0	6.0	9
company	6797.0	189.266735	131.655015	6.00	16.0	40.0	40.0	62
days_in_waiting_list	119390.0	2.321149	17.594721	0.00	0.0	0.0	0.0	0
adr	119390.0	101.831122	50.535790	-6.38	0.0	38.4	50.0	69
required_car_parking_spaces	119390.0	0.062518	0.245291	0.00	0.0	0.0	0.0	0
total_of_special_requests	119390.0	0.571363	0.792798	0.00	0.0	0.0	0.0	0

In [6]:
```
import matplotlib.pyplot as plt

# generate histograms for all the columns
df.hist(figsize=(20,15))
plt.show()
```



EDA（Week 3）

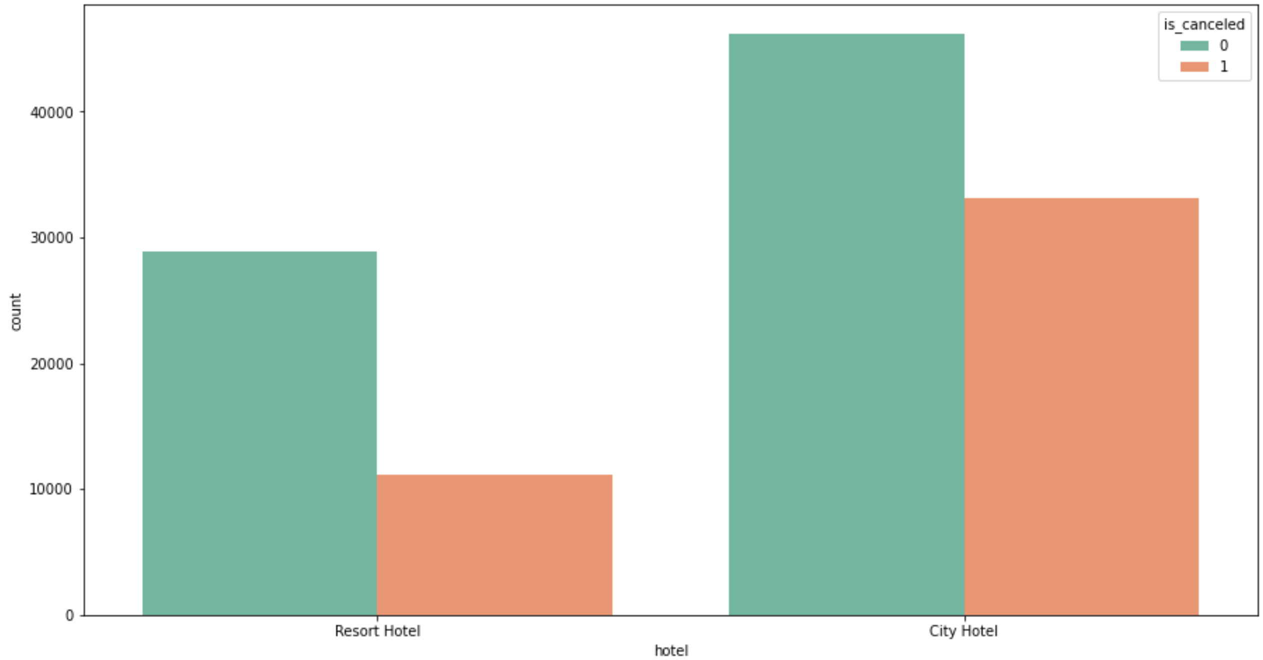
1. Hotel bookings and cancellations

In [7]:

```
# The number of hotel reservations and cancellations can directly show the actual numbe
import seaborn as sns
plt.figure(figsize=(15,8))
sns.countplot(x='hotel'
             ,data=df
             ,hue='is_canceled'
             ,palette=sns.color_palette('Set2',2)
            )
```

Out[7]:

```
<AxesSubplot:xlabel='hotel', ylabel='count'>
```



In [8]:

```
#calculate the proportion of cancellations for each unique value in the 'hotel' column
hotel_cancel=(df.loc[df['is_canceled']==1]['hotel'].value_counts()/df['hotel'].value_co
print('Hotel cancellations'.center(20),hotel_cancel,sep='\n')
```

```
Hotel cancellations
City Hotel      0.417270
Resort Hotel    0.277634
Name: hotel, dtype: float64
```

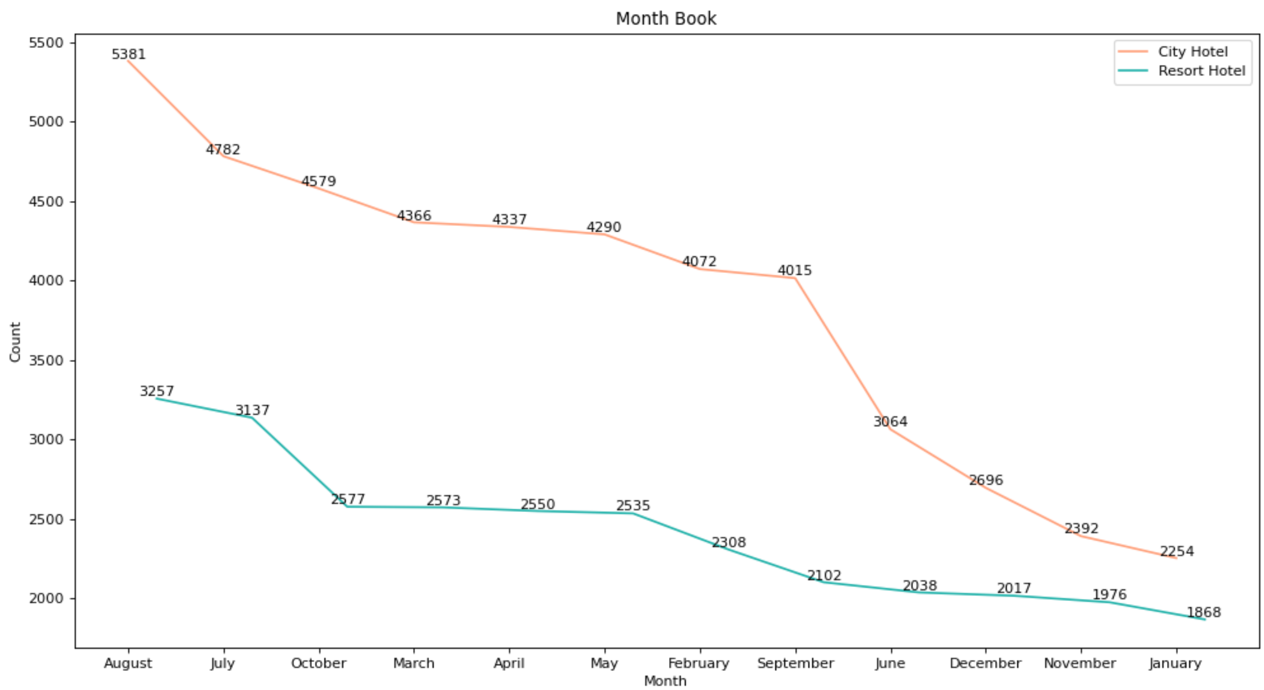
Comment: City Hotel's booking volume and cancellation volume are both higher than Resort Hotel's, but Resort Hotel's cancellation rate is 27.8%, while City Hotel's cancellation rate reaches 41.7%.

1. Hotel bookings by month

In [9]:

```
#create a plot to visualize the number of bookings for "City Hotel" and "Resort Hotel"
city_hotel=df[(df['hotel']=='City Hotel') & (df['is_canceled']==0)]
resort_hotel=df[(df['hotel']=='Resort Hotel') & (df['is_canceled']==0)]
for i in [city_hotel,resort_hotel]:
    i.index=range(i.shape[0])

city_month=city_hotel['arrival_date_month'].value_counts()
resort_month=resort_hotel['arrival_date_month'].value_counts()
name=resort_month.index
x=list(range(len(city_month.index)))
y=city_month.values
x1=[i+0.3 for i in x]
y1=resort_month.values
width=0.3
plt.figure(figsize=(15,8),dpi=80)
plt.plot(x,y,label='City Hotel',color='lightsalmon')
plt.plot(x1,y1,label='Resort Hotel',color='lightseagreen')
plt.xticks(x,name)
plt.legend()
plt.xlabel('Month')
```

```
plt.ylabel('Count')
plt.title('Month Book')
for x,y in zip(x,y):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')

for x,y in zip(x1,y1):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')
```



Comment: Peak booking months are August and July. Preliminary judgment is that the long holiday caused the peak period.

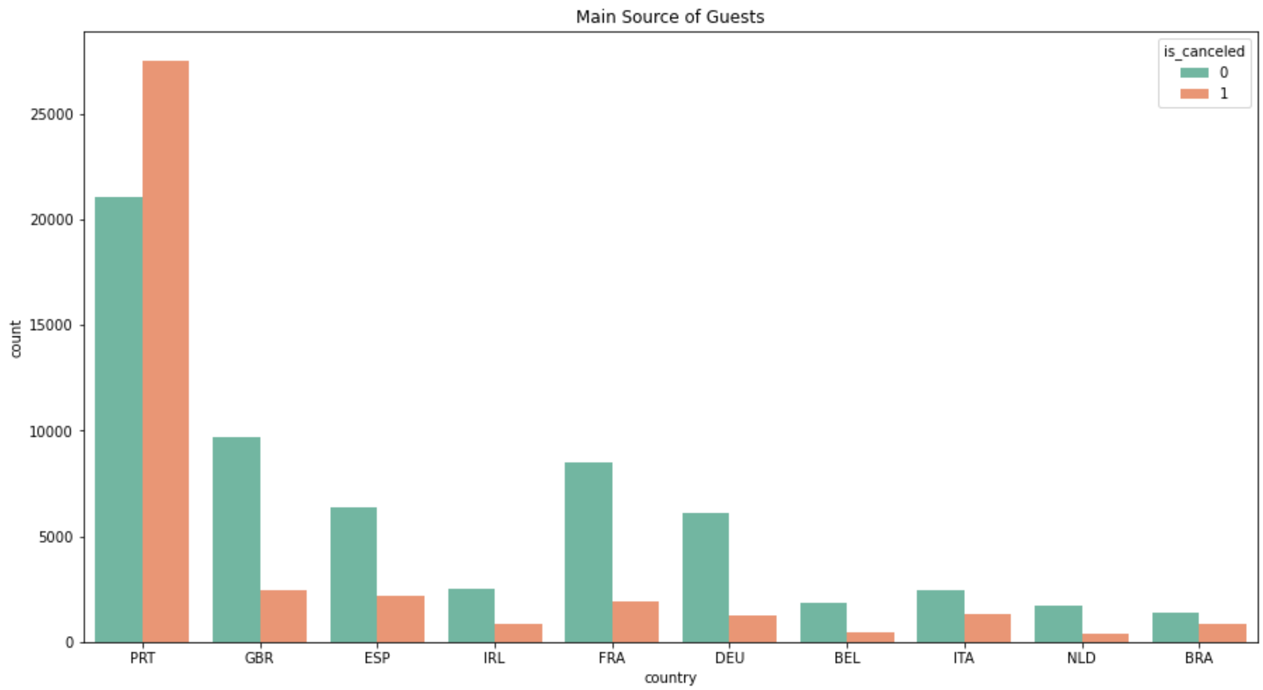
1. Customer origin and booking cancellation rate

In [10]:

```
#create a plot using Seaborn's countplot to visualize the top 10 countries from which b
country_book=df['country'].value_counts()[:10]
country_cancel=df[(df.country.isin (country_book.index)) & (df.is_canceled==1)]['countr
plt.figure(figsize=(15,8))
sns.countplot(x='country'
              ,data=df[df.country.isin (country_book.index)]
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('Main Source of Guests')
```

Out[10]:

```
Text(0.5, 1.0, 'Main Source of Guests')
```



In [11]:

```
#calculate the cancellation rate for each of the top 10 countries (those with the highe
country_cancel_rate=(country_cancel/country_book).sort_values(ascending=False)
print('Customer cancellation rates by country'.center(10),country_cancel_rate,sep='\n')
```

```
Customer cancellation rates by country
PRT    0.566351
BRA    0.373201
ITA    0.353956
ESP    0.254085
IRL    0.246519
BEL    0.202391
GBR    0.202243
FRA    0.185694
NLD    0.183935
DEU    0.167147
Name: country, dtype: float64
```

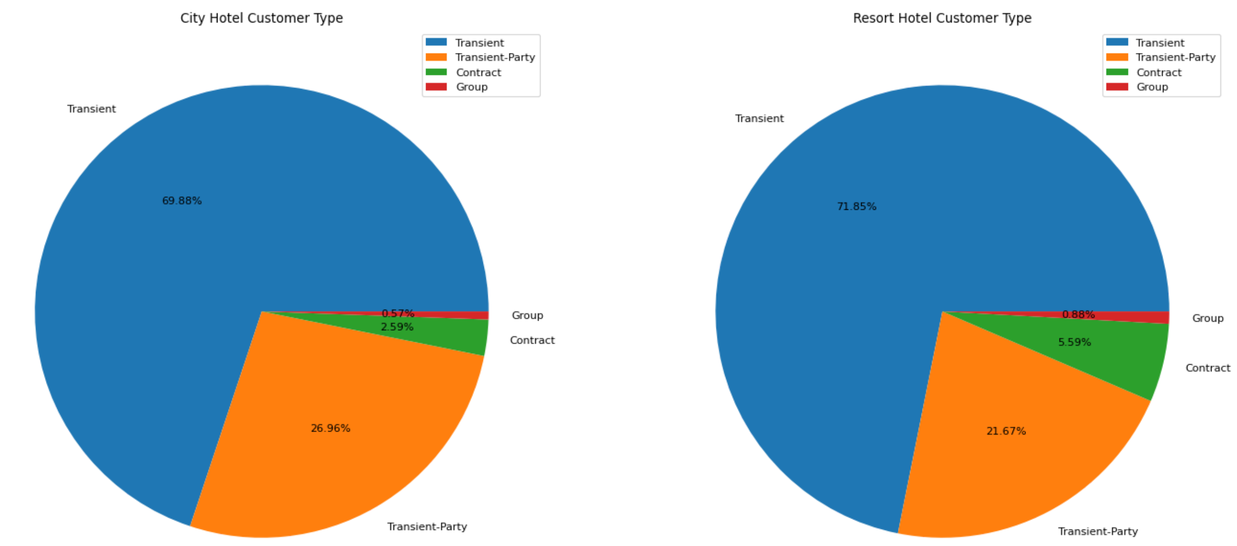
The peak season for both Resort hotel and City hotel is July and August in summer, and the main sources of tourists are European countries. This is in line with the characteristics of European tourists who prefer summer travel. It is necessary to focus on countries with high cancellation rates such as Portugal (PRT) and the United Kingdom (BRT). Main source of customers.

1. Customer type

In [12]:

```
#visualize the distribution of customer types for two types of hotels: City Hotel and R
city_customer=city_hotel.customer_type.value_counts()
resort_customer=resort_hotel.customer_type.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_customer,labels=city_customer.index,autopct='%.2f%%')
plt.legend(loc=1)
plt.title('City Hotel Customer Type')
plt.subplot(1,2,2)
plt.pie(resort_customer,labels=resort_customer.index,autopct='%.2f%%')
```

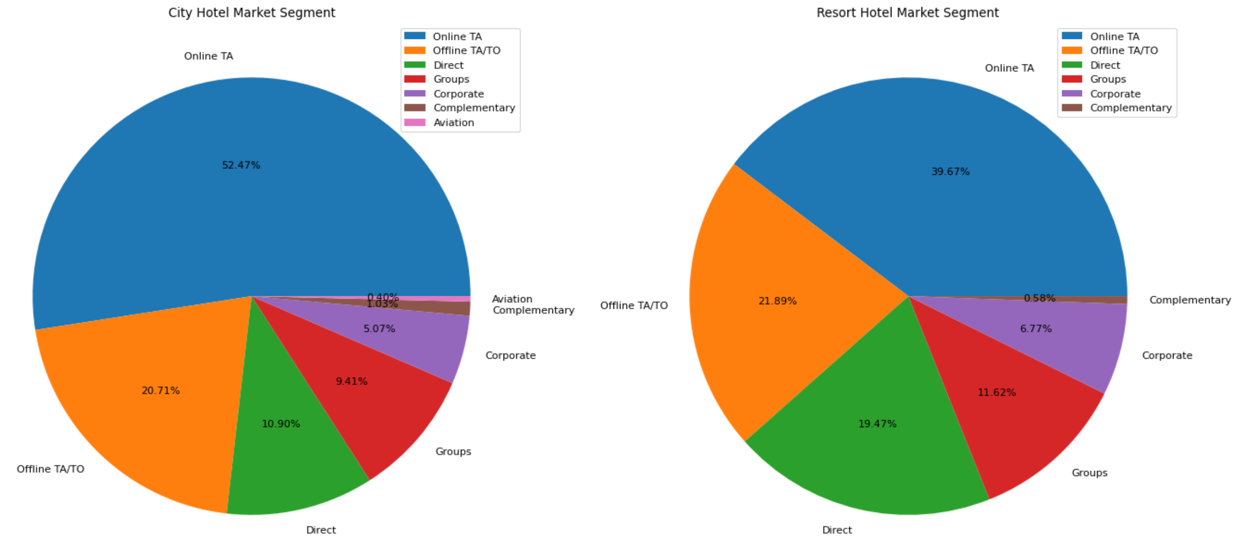
```
plt.title('Resort Hotel Customer Type')
plt.legend()
plt.show()
```



The main customer type of the hotel is transient travelers, accounting for about 70%.

1. Hotel booking method

In [13]:
```
#create pie charts to visualize the distribution of market segments for both City Hotel
city_segment=city_hotel.market_segment.value_counts()
resort_segment=resort_hotel.market_segment.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_segment,labels=city_segment.index,autopct='%.2f%%')
plt.legend()
plt.title('City Hotel Market Segment')
plt.subplot(1,2,2)
plt.pie(resort_segment,labels=resort_segment.index,autopct='%.2f%%')
plt.title('Resort Hotel Market Segment')
plt.legend()
plt.show()
```

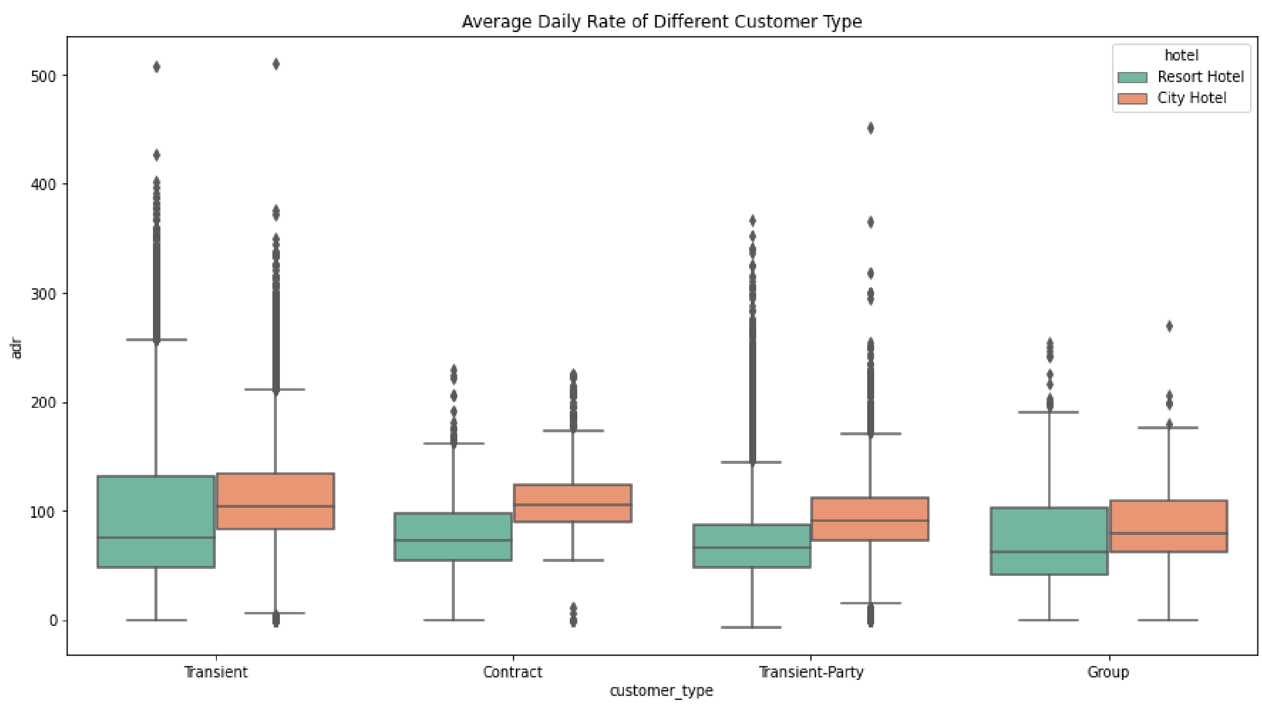


The customers of the two hotels mainly come from online travel agencies, which account for even more than 50% of the City Hotel; offline travel agencies come next, accounting for about 20%.

1. Average daily expenses of various types of passengers

In [14]:
```
#visualize the distribution of Average Daily Rate (adr) across different customer types
plt.figure(figsize=(15,8))
sns.boxplot(x='customer_type'
            ,y='adr'
            ,hue='hotel'
            ,data=df[df.is_canceled==0]
            ,palette=sns.color_palette('Set2',2)
           )
plt.title('Average Daily Rate of Different Customer Type')
```

Out[14]: Text(0.5, 1.0, 'Average Daily Rate of Different Customer Type')

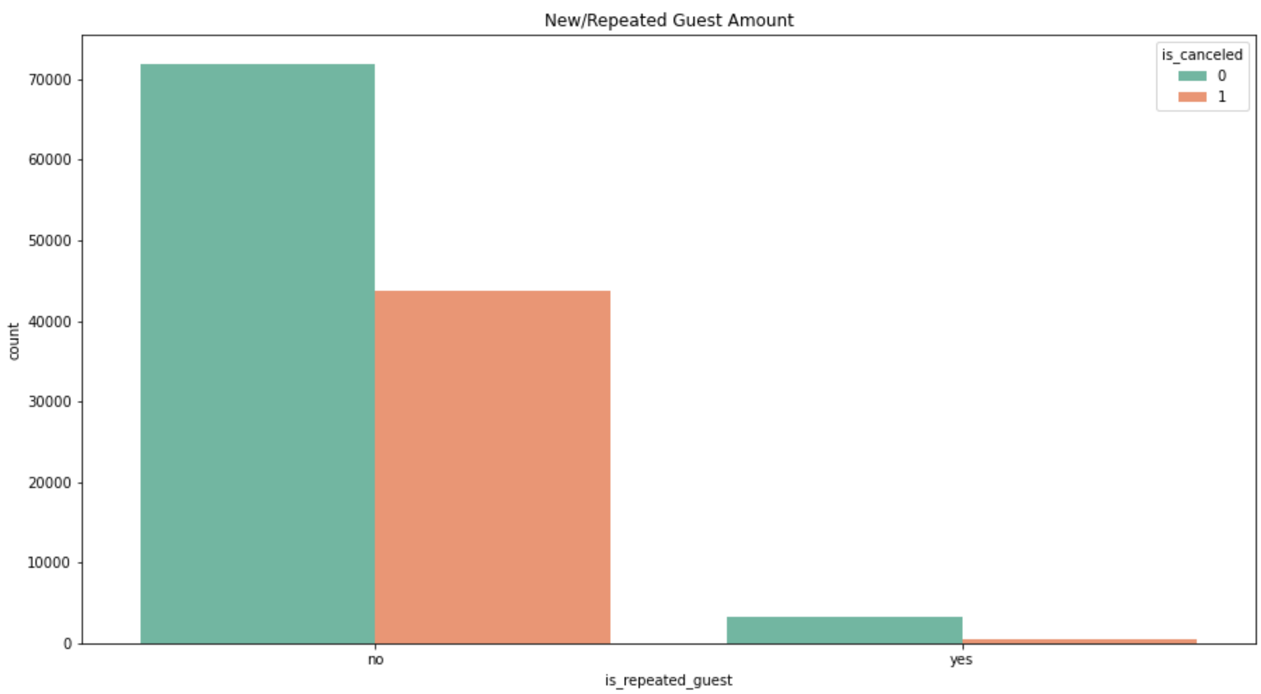


The average daily expenditure of all types of customers of City Hotel is higher than that of Resort Hotel; among the four types of customers, the consumption of individual travelers (Transient) is the highest and that of group travelers (Group) is the lowest.

7.Number of new and old customers and cancellation rate

In [15]:
```
# visualize the count of bookings, categorized by whether the guest is a repeated guest
plt.figure(figsize=(15,8))
sns.countplot(x='is_repeated_guest'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('New/Repeated Guest Amount')
plt.xticks(range(2),['no','yes'])
```

Out[15]:
```
([<matplotlib.axis.XTick at 0x2ed26e6f970>,
  <matplotlib.axis.XTick at 0x2ed26e6f940>],
 [Text(0, 0, 'no'), Text(1, 0, 'yes')])
```



In [16]:
```
#calculate and printing the cancellation rates for new and repeated guests
guest_cancel=(df.loc[df['is_canceled']==1]['is_repeated_guest'].value_counts()/df['is_r
guest_cancel.index=['New Guest', 'Repeated Guest']
print('Cancellation rate for new and old customers'.center(15),guest_cancel,sep='\n')
```

```
Cancellation rate for new and old customers
New Guest         0.377851
Repeated Guest    0.144882
Name: is_repeated_guest, dtype: float64
```

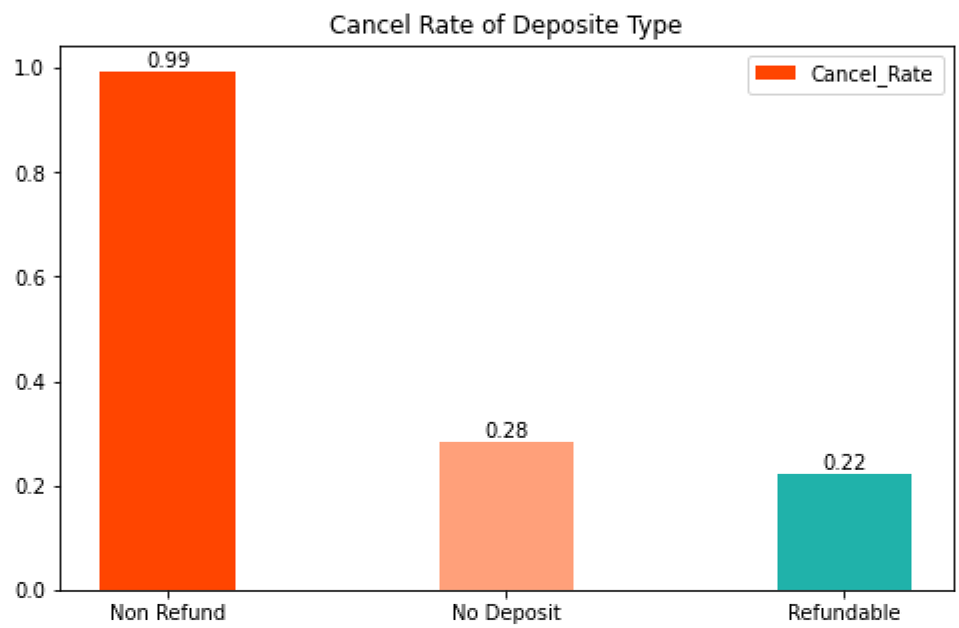
The cancellation rate for regular customers was 14.4%, while the cancellation rate for new customers reached 37.8%, which was 24 percentage points higher than that for regular customers.

1. Deposit method and reservation cancellation rate

In [17]:
```
print('Three deposit methods for booking quantity'.center(15),df['deposit_type'].value_
```

```
Three deposit methods for booking quantity
No Deposit    104641
Non Refund     14587
Refundable       162
Name: deposit_type, dtype: int64
```

In [18]:
```
#calculate the cancellation rates based on the 'deposit_type', and visualizing these ra
deposit_cancel=(df.loc[df['is_canceled']==1]['deposit_type'].value_counts()/df['deposit
plt.figure(figsize=(8,5))
x=range(len(deposit_cancel.index))
y=deposit_cancel.values
plt.bar(x,y,label='Cancel_Rate',color=['orangered','lightsalmon','lightseagreen'],width
plt.xticks(x,deposit_cancel.index)
plt.legend()
plt.title('Cancel Rate of Deposite Type')
for x,y in zip(x,y):
    plt.text(x,y,'%.2f' % y,ha = 'center',va = 'bottom')
```

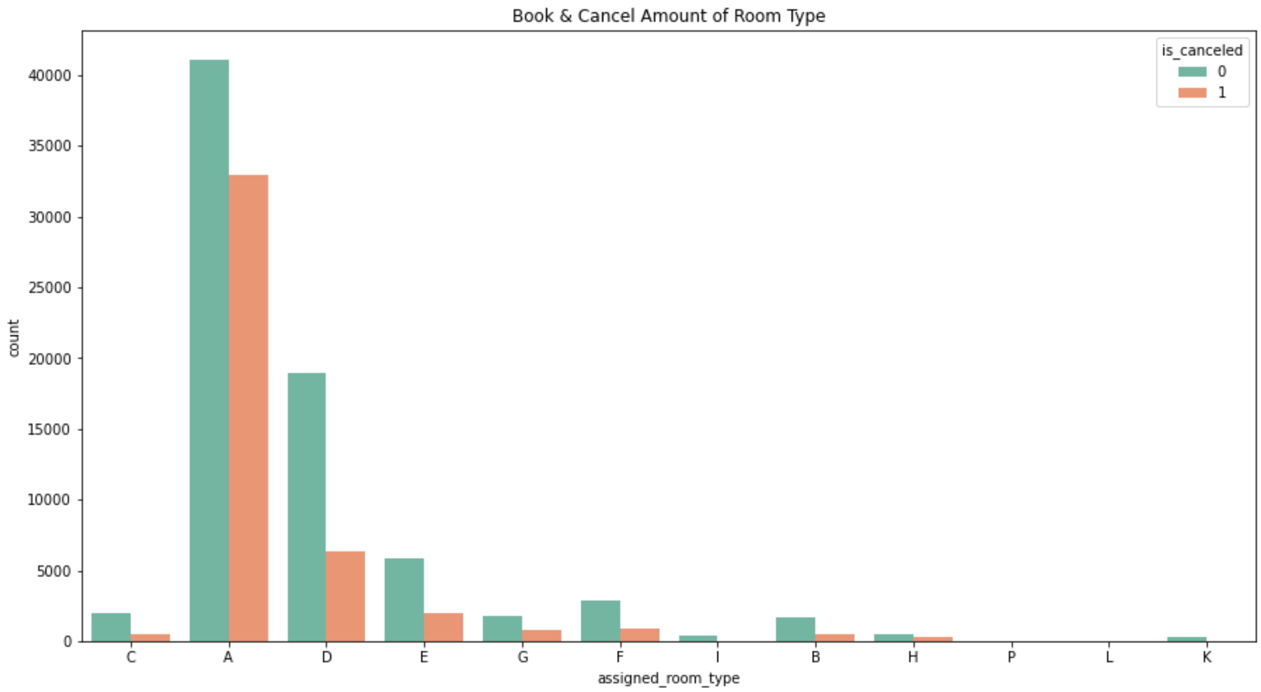


'No Deposit' is the method with the highest number of bookings and has a low cancellation rate, while the cancellation rate of non-refundable type is as high as 99%. This type of deposit method can be reduced to reduce Customer cancellation rate.

1. Room type and cancellation volume

In [19]:
```
#visualize the counts of bookings and cancellations across different assigned room type
plt.figure(figsize=(15,8))
sns.countplot(x='assigned_room_type'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('Book & Cancel Amount of Room Type')
```

Out[19]:
```
Text(0.5, 1.0, 'Book & Cancel Amount of Room Type')
```



In [20]:
```
#calculate cancellation rates for the top 7 assigned room types and printing them in de
room_cancel=df.loc[df['is_canceled']==1]['assigned_room_type'].value_counts()[:7]/df['a
print('Cancellation rates for different room types'.center(5),room_cancel.sort_values(a
```

```
Cancellation rates for different room types
A    0.444925
G    0.305523
E    0.252114
D    0.251244
F    0.247134
B    0.236708
C    0.187789
Name: assigned_room_type, dtype: float64
```

Among the top seven room types with the most bookings, the cancellation rates of room types A and G are higher than other room types, and the cancellation rate of room type A is as high as 44.5%.

Conclusion

1. The booking volume and cancellation rate of City Hotel are much higher than that of Resort Hotel. The hotel should conduct customer surveys to gain an in-depth understanding of the factors that cause customers to give up on bookings in order to reduce customer cancellation rates.

2. Hotels should make good use of the peak tourist season of July and August every year. They can increase prices appropriately while ensuring service quality to obtain more profits, and conduct preferential activities during the off-season (winter), such as Christmas sales and New Year activities, to reduce Hotel vacancy rate.

3. Hotels need to analyze customer profiles from major source countries such as Portugal and the United Kingdom, understand the attribute tags, preferences and consumption characteristics of these customers, and launch exclusive services to reduce customer cancellation rates.
4. Since individual travelers are the main customer group of hotels and have high consumption levels, hotels can increase the promotion and marketing of independent travelers through online and offline travel agencies, thereby attracting more tourists of this type.
5. The cancellation rate of new customers is 24% higher than that of old customers. Therefore, hotels should focus on the booking and check-in experience of new customers, and provide more guidance and benefits to new customers, such as providing discounts to first-time customers and conducting research on new customers. Provide feedback on satisfaction and dissatisfaction with your stay to improve future services and maintain good old customers.
6. The cancellation rate of non-refundable deposits is as high as 99%. Hotels should optimize this method, such as returning 50% of the deposit, or cancel this method directly to increase the occupancy rate.
7. The cancellation rate of room types A and G is much higher than that of other room types. The hotel should carefully confirm the room information with the customer when making a reservation, so that the customer can fully understand the room situation, avoid cognitive errors, and at the same time be able to understand the room facilities. Optimize and improve service levels.

Data Processing (Week 4)

In [21]:

```
#create a new DataFrame 'df1' from 'df'
df1=df.drop(labels=['reservation_status_date'],axis=1)
```

Handling Categorical Variables

In [22]:

```
cate=df1.columns[df1.dtypes == "object"].tolist() #getting the names of all columns in
#categorical variables expressed as numbers
num_cate=['agent','company','is_repeated_guest']
cate=cate+num_cate
```

In [23]:

```
import numpy as np #linear algebra
#creating a dictionary
results={}
for i in ['agent','company']:
    result=np.sort(df1[i].unique())
    results[i]=result
results
```

Out[23]:

```
{'agent': array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,  11.,
         12.,  13.,  14.,  15.,  16.,  17.,  19.,  20.,  21.,  22.,  23.,
         24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,  33.,  34.,
         35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  44.,  45.,  47.,
```

```
        50.,  52.,  53.,  54.,  55.,  56.,  57.,  58.,  59.,  60.,  61.,
        63.,  64.,  66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,
        75.,  77.,  78.,  79.,  81.,  82.,  83.,  85.,  86.,  87.,  88.,
        89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  98.,  99., 103.,
       104., 105., 106., 107., 110., 111., 112., 114., 115., 117., 118.,
       119., 121., 122., 126., 127., 128., 129., 132., 133., 134., 135.,
       138., 139., 141., 142., 143., 144., 146., 147., 148., 149., 150.,
       151., 152., 153., 154., 155., 156., 157., 158., 159., 162., 163.,
       165., 167., 168., 170., 171., 173., 174., 175., 177., 179., 180.,
       181., 182., 183., 184., 185., 187., 191., 192., 193., 195., 196.,
       197., 201., 205., 208., 210., 211., 213., 214., 215., 216., 219.,
       220., 223., 227., 229., 232., 234., 235., 236., 240., 241., 242.,
       243., 244., 245., 247., 248., 249., 250., 251., 252., 253., 254.,
       256., 257., 258., 261., 262., 265., 267., 269., 270., 273., 275.,
       276., 278., 280., 281., 282., 283., 285., 286., 287., 288., 289.,
       290., 291., 294., 295., 296., 298., 299., 300., 301., 302., 303.,
       304., 305., 306., 307., 308., 310., 313., 314., 315., 321., 323.,
       324., 325., 326., 327., 328., 330., 331., 332., 333., 334., 335.,
       336., 337., 339., 341., 344., 346., 348., 350., 352., 354., 355.,
       358., 359., 360., 363., 364., 367., 368., 370., 371., 375., 378.,
       384., 385., 387., 388., 390., 391., 393., 394., 397., 403., 404.,
       405., 406., 408., 410., 411., 414., 416., 418., 420., 423., 425.,
       426., 427., 429., 430., 431., 432., 433., 434., 436., 438., 440.,
       441., 444., 446., 449., 450., 451., 453., 454., 455., 459., 461.,
       464., 467., 468., 469., 472., 474., 475., 476., 479., 480., 481.,
       483., 484., 492., 493., 495., 497., 502., 508., 509., 510., 526.,
       527., 531., 535.,  nan]),
 'company': array([  6.,   8.,   9.,  10.,  11.,  12.,  14.,  16.,  18.,  20.,  22.,
        28.,  29.,  31.,  32.,  34.,  35.,  37.,  38.,  39.,  40.,  42.,
        43.,  45.,  46.,  47.,  48.,  49.,  51.,  52.,  53.,  54.,  59.,
        61.,  62.,  64.,  65.,  67.,  68.,  71.,  72.,  73.,  76.,  77.,
        78.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  88.,  91.,  92.,
        93.,  94.,  96.,  99., 100., 101., 102., 103., 104., 105., 106.,
       107., 108., 109., 110., 112., 113., 115., 116., 118., 120., 122.,
       126., 127., 130., 132., 135., 137., 139., 140., 142., 143., 144.,
       146., 148., 149., 150., 153., 154., 158., 159., 160., 163., 165.,
       167., 168., 169., 174., 178., 179., 180., 183., 184., 185., 186.,
       192., 193., 195., 197., 200., 202., 203., 204., 207., 209., 210.,
       212., 213., 215., 216., 217., 218., 219., 220., 221., 222., 223.,
       224., 225., 227., 229., 230., 232., 233., 234., 237., 238., 240.,
       242., 243., 245., 246., 250., 251., 253., 254., 255., 257., 258.,
       259., 260., 263., 264., 268., 269., 270., 271., 272., 273., 274.,
       275., 277., 278., 279., 280., 281., 282., 284., 286., 287., 288.,
       289., 290., 291., 292., 293., 297., 301., 302., 304., 305., 307.,
       308., 309., 311., 312., 313., 314., 316., 317., 318., 319., 320.,
       321., 323., 324., 325., 329., 330., 331., 332., 333., 334., 337.,
       338., 341., 342., 343., 346., 347., 348., 349., 350., 351., 352.,
       353., 355., 356., 357., 358., 360., 361., 362., 364., 365., 366.,
       367., 368., 369., 370., 371., 372., 373., 376., 377., 378., 379.,
       380., 382., 383., 384., 385., 386., 388., 390., 391., 392., 393.,
       394., 395., 396., 397., 398., 399., 400., 401., 402., 403., 405.,
       407., 408., 409., 410., 411., 412., 413., 415., 416., 417., 418.,
       419., 420., 421., 422., 423., 424., 425., 426., 428., 429., 433.,
       435., 436., 437., 439., 442., 443., 444., 445., 446., 447., 448.,
       450., 451., 452., 454., 455., 456., 457., 458., 459., 460., 461.,
       465., 466., 470., 477., 478., 479., 481., 482., 483., 484., 485.,
       486., 487., 489., 490., 491., 492., 494., 496., 497., 498., 499.,
       501., 504., 506., 507., 511., 512., 513., 514., 515., 516., 518.,
       520., 521., 523., 525., 528., 530., 531., 534., 539., 541., 543.,
        nan])}
```

In [24]:
```
# the agent and company columns have a large number of empty values and no 0 values, so
df1[['agent','company']]=df1[['agent','company']].fillna(0,axis=0)
```

In [25]:
```
df1.loc[:,cate].isnull().mean()
```

Out[25]:
```
hotel                   0.000000
arrival_date_month      0.000000
meal                    0.000000
country                 0.004087
market_segment          0.000000
distribution_channel    0.000000
reserved_room_type      0.000000
assigned_room_type      0.000000
deposit_type            0.000000
customer_type           0.000000
reservation_status      0.000000
name                    0.000000
email                   0.000000
phone-number            0.000000
credit_card             0.000000
agent                   0.000000
company                 0.000000
is_repeated_guest       0.000000
dtype: float64
```

In [26]:
```
#create new variables in_company and in_agent to classify passengers. If company and ag
df1.loc[df1['company'] == 0,'in_company']='NO'
df1.loc[df1['company'] != 0,'in_company']='YES'
df1.loc[df1['agent'] == 0,'in_agent']='NO'
df1.loc[df1['agent'] != 0,'in_agent']='YES'
```

In [27]:
```
#create a new feature same_assignment. If the booked room type is consistent with the a
df1.loc[df1['reserved_room_type'] == df1['assigned_room_type'],'same_assignment']='Yes'
df1.loc[df1['reserved_room_type'] != df1['assigned_room_type'],'same_assignment']='No'
```

In [28]:
```
#delete four features except 'reserved_room_type', 'assigned_room_type', 'agent', 'comp
df1=df1.drop(labels=['reserved_room_type','assigned_room_type','agent','company'],axis=
```

In [29]:
```
#reset 'is_repeated_guest', frequent guests are marked as YES, non-repeated guests are
df1['is_repeated_guest'][df1['is_repeated_guest']==0]='NO'
df1['is_repeated_guest'][df1['is_repeated_guest']==1]='YES'
```

In [30]:
```
#filling the missing values in the 'country' column of the DataFrame 'df1' with the mod
df1['country']=df1['country'].fillna(df1['country'].mode()[0])
```

In [31]:
```
for i in ['in_company','in_agent','same_assignment']:
    cate.append(i)

for i in ['reserved_room_type','assigned_room_type','agent','company']:
```

```
    cate.remove(i)
cate
```

Out[31]:
```
['hotel',
 'arrival_date_month',
 'meal',
 'country',
 'market_segment',
 'distribution_channel',
 'deposit_type',
 'customer_type',
 'reservation_status',
 'name',
 'email',
 'phone-number',
 'credit_card',
 'is_repeated_guest',
 'in_company',
 'in_agent',
 'same_assignment']
```

In [32]:
```
#encoding categorical features
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
oe = oe.fit(df1.loc[:,cate])
df1.loc[:,cate] = oe.transform(df1.loc[:,cate])
```

Working With Continuous Variables

In [33]:
```
#to filter out continuous variables, you need to delete the label 'is_canceled' first.
col=df1.columns.tolist()
col.remove('is_canceled')
for i in cate:
    col.remove(i)
col
```

Out[33]:
```
['lead_time',
 'arrival_date_year',
 'arrival_date_week_number',
 'arrival_date_day_of_month',
 'stays_in_weekend_nights',
 'stays_in_week_nights',
 'adults',
 'children',
 'babies',
 'previous_cancellations',
 'previous_bookings_not_canceled',
 'booking_changes',
 'days_in_waiting_list',
 'adr',
 'required_car_parking_spaces',
 'total_of_special_requests']
```

In [34]:
```
df1[col].isnull().sum()
```

Out[34]:
```
lead_time                    0
arrival_date_year            0
```

```
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          4
babies                            0
previous_cancellations            0
previous_bookings_not_canceled    0
booking_changes                   0
days_in_waiting_list              0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
dtype: int64
```

In [35]:

```
#use mode to fill null values in xtrain children column
df1['children']=df1['children'].fillna(df1['children'].mode()[0])
```

In [36]:

```
#continuous variables are dimensionless
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss = ss.fit(df1.loc[:,col])
df1.loc[:,col] = ss.transform(df1.loc[:,col])
```

Correlation Coefficient of Each Variable

In [37]:

```
#calculating the correlation of all numerical columns with the 'is_canceled column' in
cor=df1.corr()
cor=abs(cor['is_canceled']).sort_values()
cor
```

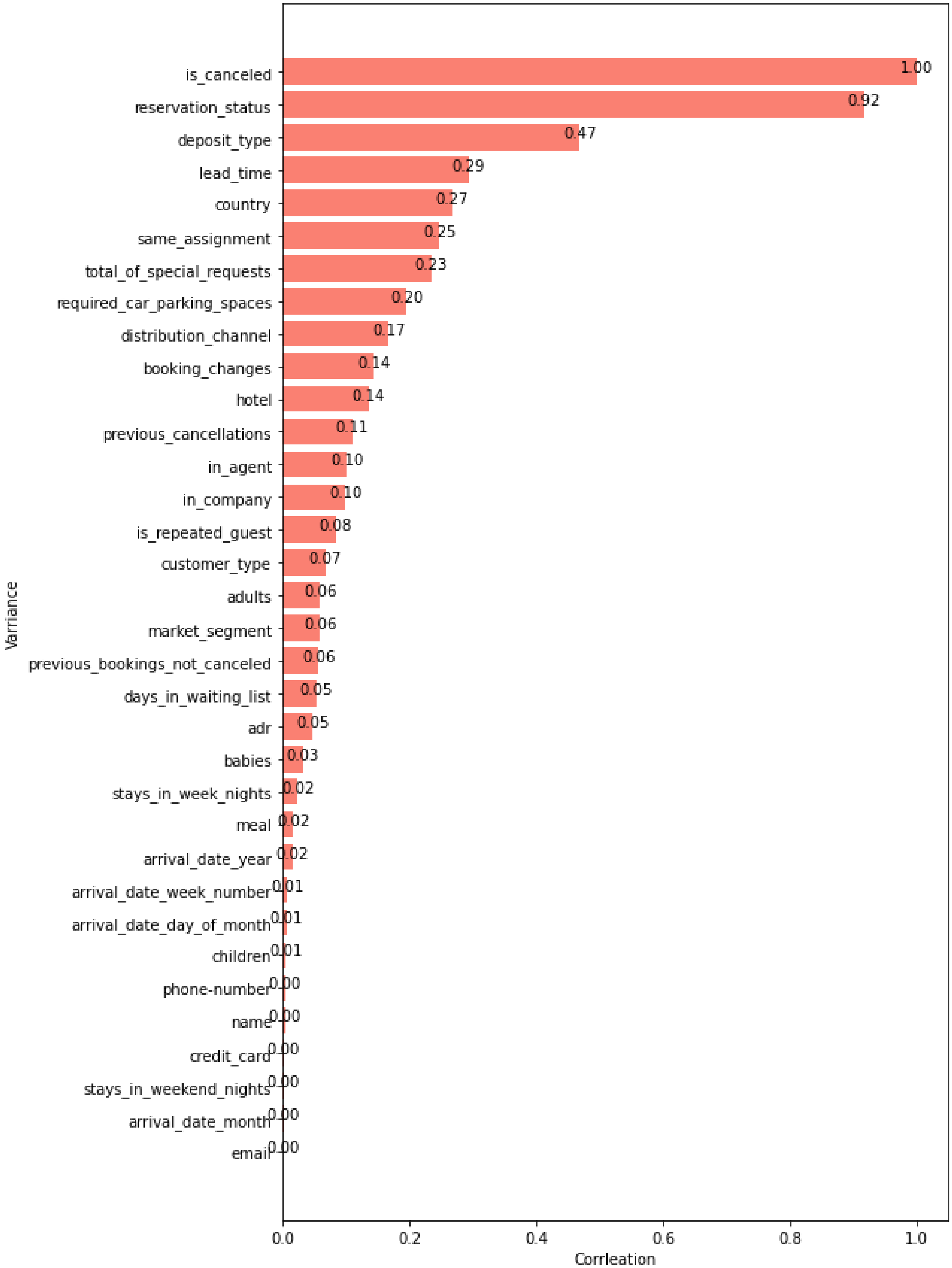
Out[37]:

```
email                             0.000723
arrival_date_month                0.001491
stays_in_weekend_nights           0.001791
credit_card                       0.002515
name                              0.004253
phone-number                      0.004342
children                          0.005036
arrival_date_day_of_month         0.006130
arrival_date_week_number          0.008148
arrival_date_year                 0.016660
meal                              0.017678
stays_in_week_nights              0.024765
babies                            0.032491
adr                               0.047557
days_in_waiting_list              0.054186
previous_bookings_not_canceled    0.057358
market_segment                    0.059338
adults                            0.060017
customer_type                     0.068140
is_repeated_guest                 0.084793
in_company                        0.099310
in_agent                          0.102068
previous_cancellations            0.110133
hotel                             0.136531
```

```
booking_changes                0.144381
distribution_channel           0.167600
required_car_parking_spaces    0.195498
total_of_special_requests      0.234658
same_assignment                0.247770
country                        0.267502
lead_time                      0.293123
deposit_type                   0.468634
reservation_status             0.917196
is_canceled                    1.000000
Name: is_canceled, dtype: float64
```

In [38]:

```
#create a horizontal bar plot using Matplotlib to visualize the absolute correlation va
plt.figure(figsize=(8,15))
x=range(len(cor.index))
name=cor.index
y=abs(cor.values)
plt.barh(x,y,color='salmon')
plt.yticks(x,name)
for x,y in zip(x,y):
    plt.text(y,x-0.1,'%.2f' % y,ha = 'center',va = 'bottom')
plt.xlabel('Corrleation')
plt.ylabel('Varriance')
plt.show()
```



The reservation status ('reservation_status') has the highest correlation with whether to cancel the reservation, reaching 0.92, but considering that it may cause the model to overfit in the future, it is deleted; the deposit type ('deposit_type') reaches 0.47, creating a characteristic Whether the reservation and assigned room type are consistent ('same_assignment') also has a correlation of 0.25.

In [39]:
```
#copy 'df1' with the column labeled 'reservation_status' dropped.
df2=df1.drop('reservation_status',axis=1)
```

Week 5

In [40]:
```
#dropping columns that are not useful
useless_col = ['email', 'phone-number', 'credit_card', 'name', 'days_in_waiting_list',
               'reservation_status', 'country', 'days_in_waiting_list']

df.drop(useless_col, axis = 1, inplace = True)
```

In [41]:
```
df.head()
```

Out[41]:

	hotel	is_canceled	lead_time	arrival_date_month	arrival_date_week_number	arrival_date_day_of_mont
0	Resort Hotel	0	342	July	27	
1	Resort Hotel	0	737	July	27	
2	Resort Hotel	0	7	July	27	
3	Resort Hotel	0	13	July	27	
4	Resort Hotel	0	14	July	27	

5 rows × 26 columns

In [42]:
```
# creating numerical and categorical dataframes
cat_cols = [col for col in df.columns if df[col].dtype == 'O']
cat_cols
```

Out[42]:
```
['hotel',
 'arrival_date_month',
 'meal',
 'market_segment',
 'distribution_channel',
 'reserved_room_type',
 'deposit_type',
 'customer_type',
 'reservation_status_date']
```

In [43]:
```
cat_df = df[cat_cols]
cat_df.head()
```

Out[43]:

	hotel	arrival_date_month	meal	market_segment	distribution_channel	reserved_room_type	deposit_
0	Resort Hotel	July	BB	Direct	Direct	C	No De
1	Resort Hotel	July	BB	Direct	Direct	C	No De
2	Resort Hotel	July	BB	Direct	Direct	A	No De
3	Resort Hotel	July	BB	Corporate	Corporate	A	No De
4	Resort Hotel	July	BB	Online TA	TA/TO	A	No De

In [44]:

```
#Convert 'reservation_status_date' to DateTime type
cat_df['reservation_status_date'] = pd.to_datetime(cat_df['reservation_status_date'])
#Extract the Year from the 'reservation_status_date'
cat_df['year'] = cat_df['reservation_status_date'].dt.year
#Extract the Month from the 'reservation_status_date'
cat_df['month'] = cat_df['reservation_status_date'].dt.month
#Extract the Day from the 'reservation_status_date'
cat_df['day'] = cat_df['reservation_status_date'].dt.day
```

In [45]:

```
cat_df.drop(['reservation_status_date','arrival_date_month'] , axis = 1, inplace = True
```

In [46]:

```
cat_df.head(15)
```

Out[46]:

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_typ
0	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
1	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
2	Resort Hotel	BB	Direct	Direct	A	No Deposit	Transier
3	Resort Hotel	BB	Corporate	Corporate	A	No Deposit	Transier
4	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
5	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
6	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
7	Resort Hotel	FB	Direct	Direct	C	No Deposit	Transier

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_typ
8	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
9	Resort Hotel	HB	Offline TA/TO	TA/TO	D	No Deposit	Transier
10	Resort Hotel	BB	Online TA	TA/TO	E	No Deposit	Transier
11	Resort Hotel	HB	Online TA	TA/TO	D	No Deposit	Transier
12	Resort Hotel	BB	Online TA	TA/TO	D	No Deposit	Transier
13	Resort Hotel	HB	Online TA	TA/TO	G	No Deposit	Transier
14	Resort Hotel	BB	Online TA	TA/TO	E	No Deposit	Transier

In [47]:

```
# printing unique values of each column
for col in cat_df.columns:
    print(f"{col}: \n{cat_df[col].unique()}\n")
```

```
hotel:
['Resort Hotel' 'City Hotel']

meal:
['BB' 'FB' 'HB' 'SC' 'Undefined']

market_segment:
['Direct' 'Corporate' 'Online TA' 'Offline TA/TO' 'Complementary' 'Groups'
 'Undefined' 'Aviation']

distribution_channel:
['Direct' 'Corporate' 'TA/TO' 'Undefined' 'GDS']

reserved_room_type:
['C' 'A' 'D' 'E' 'G' 'F' 'H' 'L' 'P' 'B']

deposit_type:
['No Deposit' 'Refundable' 'Non Refund']

customer_type:
['Transient' 'Contract' 'Transient-Party' 'Group']

year:
[2015 2014 2016 2017]

month:
[ 7  5  4  6  3  8  9  1 11 10 12  2]

day:
[ 1  2  3  6 22 23  5  7  8 11 15 16 29 19 18  9 13  4 12 26 17 10 20 14
 30 28 25 21 27 24 31]
```

In [48]:

```
# encoding categorical variables, which can be in text/string format, into numerical fo

cat_df['hotel'] = cat_df['hotel'].map({'Resort Hotel' : 0, 'City Hotel' : 1})

cat_df['meal'] = cat_df['meal'].map({'BB' : 0, 'FB': 1, 'HB': 2, 'SC': 3, 'Undefined':

cat_df['market_segment'] = cat_df['market_segment'].map({'Direct': 0, 'Corporate': 1, '
                                                           'Complementary': 4, 'Groups'

cat_df['distribution_channel'] = cat_df['distribution_channel'].map({'Direct': 0, 'Corp
                                                                       'GDS': 4})

cat_df['reserved_room_type'] = cat_df['reserved_room_type'].map({'C': 0, 'A': 1, 'D': 2
                                                                   'L': 7, 'B': 8})

cat_df['deposit_type'] = cat_df['deposit_type'].map({'No Deposit': 0, 'Refundable': 1,

cat_df['customer_type'] = cat_df['customer_type'].map({'Transient': 0, 'Contract': 1, '

cat_df['year'] = cat_df['year'].map({2015: 0, 2014: 1, 2016: 2, 2017: 3})
```

In [49]:

```
cat_df.head(15)
```

Out[49]:

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_type
0	0	0	0	0	0.0	0	(
1	0	0	0	0	0.0	0	(
2	0	0	0	0	1.0	0	(
3	0	0	1	1	1.0	0	(
4	0	0	2	2	1.0	0	(
5	0	0	2	2	1.0	0	(
6	0	0	0	0	0.0	0	(
7	0	1	0	0	0.0	0	(
8	0	0	2	2	1.0	0	(
9	0	2	3	2	2.0	0	(
10	0	0	2	2	3.0	0	(
11	0	2	2	2	2.0	0	(
12	0	0	2	2	2.0	0	(
13	0	2	2	2	4.0	0	(
14	0	0	2	2	3.0	0	(

In [50]:

```
num_df = df.drop(columns = cat_cols, axis = 1)
num_df.drop('is_canceled', axis = 1, inplace = True)
num_df
```

Out[50]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_
0	342	27	1	0	
1	737	27	1	0	
2	7	27	1	0	
3	13	27	1	0	
4	14	27	1	0	
...	...	...	...	...	
119385	23	35	30	2	
119386	102	35	31	2	
119387	34	35	31	2	
119388	109	35	31	2	
119389	205	35	29	2	

119390 rows × 16 columns

In [51]:
```
num_df.var()
```

Out[51]:
```
lead_time                          11419.721511
arrival_date_week_number             185.099790
arrival_date_day_of_month             77.102966
stays_in_weekend_nights                0.997229
stays_in_week_nights                   3.641554
adults                                 0.335543
children                               0.158851
babies                                 0.009494
is_repeated_guest                      0.030894
previous_cancellations                 0.712904
previous_bookings_not_canceled         2.242317
agent                              12271.000405
company                            17333.042879
adr                                 2553.866100
required_car_parking_spaces            0.060168
total_of_special_requests              0.628529
dtype: float64
```

In [52]:
```
# normalizing numerical variables, uses the natural logarithm to transform the data.
#It's essential to add 1 before taking the log to handle instances where the column val
num_df['lead_time'] = np.log(num_df['lead_time'] + 1)
num_df['arrival_date_week_number'] = np.log(num_df['arrival_date_week_number'] + 1)
num_df['arrival_date_day_of_month'] = np.log(num_df['arrival_date_day_of_month'] + 1)
num_df['agent'] = np.log(num_df['agent'] + 1)
num_df['company'] = np.log(num_df['company'] + 1)
num_df['adr'] = np.log(num_df['adr'] + 1)
```

In [53]:
```
num_df.var()
```

Out[53]:
```
lead_time                          2.591420
arrival_date_week_number           0.441039
arrival_date_day_of_month          0.506267
stays_in_weekend_nights            0.997229
stays_in_week_nights               3.641554
adults                             0.335543
children                           0.158851
babies                             0.009494
is_repeated_guest                  0.030894
previous_cancellations             0.712904
previous_bookings_not_canceled     2.242317
agent                              2.536204
company                            0.755665
adr                                0.540353
required_car_parking_spaces        0.060168
total_of_special_requests          0.628529
dtype: float64
```

In [54]:
```
num_df['adr'] = num_df['adr'].fillna(value = num_df['adr'].mean())
num_df.head(15)
```

Out[54]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_we
0	5.837730	3.332205	0.693147	0	
1	6.603944	3.332205	0.693147	0	
2	2.079442	3.332205	0.693147	0	
3	2.639057	3.332205	0.693147	0	
4	2.708050	3.332205	0.693147	0	
5	2.708050	3.332205	0.693147	0	
6	0.000000	3.332205	0.693147	0	
7	2.302585	3.332205	0.693147	0	
8	4.454347	3.332205	0.693147	0	
9	4.330733	3.332205	0.693147	0	
10	3.178054	3.332205	0.693147	0	
11	3.583519	3.332205	0.693147	0	
12	4.234107	3.332205	0.693147	0	
13	2.944439	3.332205	0.693147	0	
14	3.637586	3.332205	0.693147	0	

Prepare the independent and dependent variables for a modeling task

In [55]:

```
#merging categorical and numerical dataframes
#X = pd.concat([cat_df, num_df], axis = 1)
#y = df['is_canceled']
x=df2.loc[:,df2.columns != 'is_canceled' ]
y=df2.loc[:,'is_canceled']
from sklearn.model_selection import train_test_split as tts
xtrain,xtest,ytrain,ytest=tts(x,y,test_size=0.3,random_state=90)
for i in [xtrain,xtest,ytrain,ytest]:
    i.index=range(i.shape[0])
```

In [57]:

```
x.shape, y.shape
```

Out[57]:

```
((119390, 32), (119390,))
```

In [59]:

```
# splitting data into training set and test set
#from sklearn.model_selection import train_test_split, GridSearchCV
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
```

In [60]:

```
xtrain.head()
```

Out[60]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	0.0	1.029252	1.192195	5.0	0.061361	-0.!
1	0.0	0.102829	-0.221286	8.0	-0.600156	-1.!
2	1.0	0.168334	1.192195	6.0	-0.085642	1.(
3	1.0	0.767233	1.192195	5.0	-0.012141	-0.8
4	0.0	-0.421208	-0.221286	11.0	0.943385	1.'

5 rows × 32 columns

In [61]:

```
xtest.head()
```

Out[61]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	0.0	-0.963961	-1.634768	2.0	1.898910	1.2
1	0.0	-0.861025	-0.221286	5.0	0.208365	0.3
2	0.0	1.431638	-0.221286	5.0	0.355369	1.7
3	0.0	-0.879741	-0.221286	11.0	0.722879	-1.2
4	0.0	-0.224694	-0.221286	11.0	0.943385	1.'

5 rows × 32 columns

In [62]:

```
ytrain.head(), ytest.head()
```

Out[62]:

```
(0    1
 1    0
 2    0
 3    1
 4    0
 Name: is_canceled, dtype: int64,
 0    0
 1    0
 2    0
 3    0
 4    0
 Name: is_canceled, dtype: int64)
```

Week 6

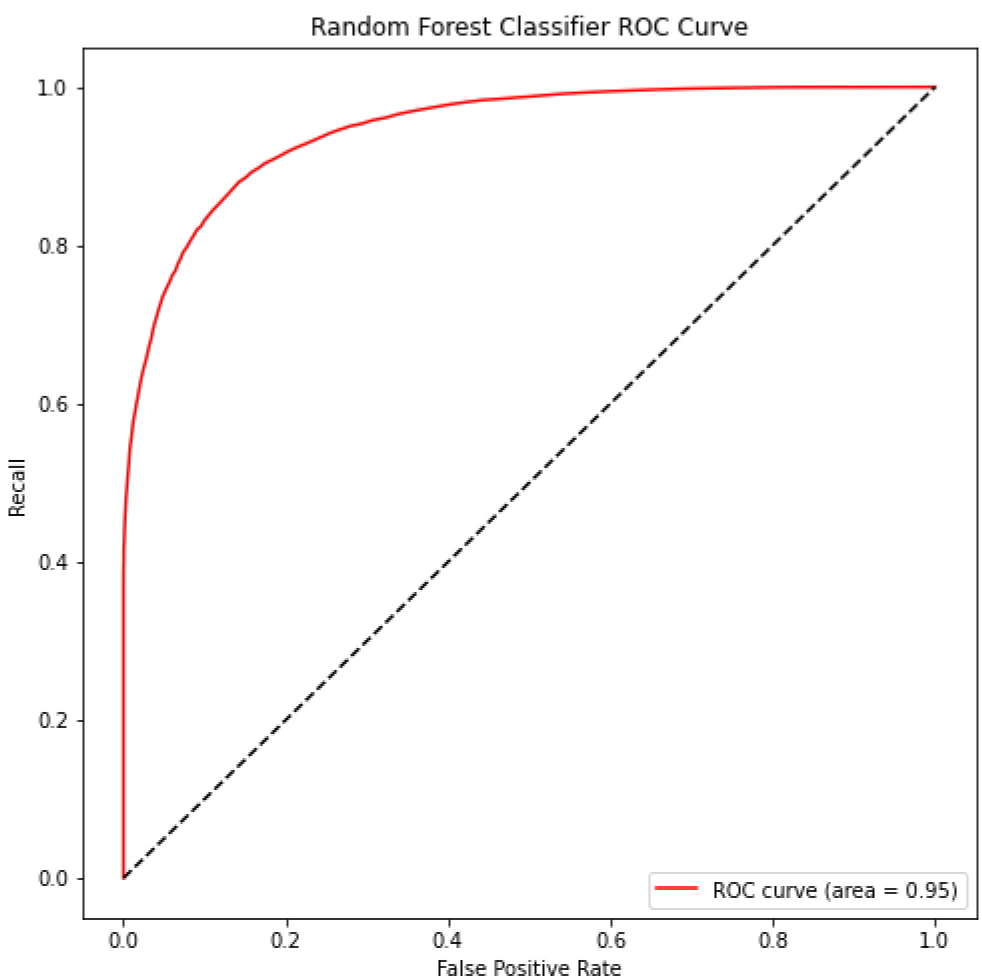
In [63]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score as cvs,KFold
from sklearn.metrics import accuracy_score
#Initialize the RandomForestClassifier with 100 estimators and a random seed
```

```
rfc=RandomForestClassifier(n_estimators=100,random_state=90)
#Define a KFold cross-validation object with 10 splits, shuffling the data, and using a
cv=KFold(n_splits=10, shuffle = True, random_state=90)
#Use cross_val_score to perform 10-fold cross-validation and calculate the mean accurac
rfc_score=cvs(rfc,xtrain,ytrain,cv=cv).mean()
rfc.fit(xtrain,ytrain)
y_score=rfc.predict_proba(xtest)[:,1]
#Generate binary predictions for the test data
rfc_pred=rfc.predict(xtest)
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score as AUC
FPR, recall, thresholds = roc_curve(ytest,y_score, pos_label=1)
rfc_auc = AUC(ytest,y_score)
```

In [64]:

```
# Draw ROC curve
plt.figure(figsize=(8,8))
plt.plot(FPR, recall, color='red',label='ROC curve (area = %0.2f)' % rfc_auc)
plt.plot([0, 1], [0, 1], color='black', linestyle='--')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('Recall')
plt.title('Random Forest Classifier ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



In [72]:
```
pip install optuna
```

```
Collecting optuna
  Downloading optuna-3.3.0-py3-none-any.whl (404 kB)
     -------------------------------------- 404.2/404.2 kB 5.1 MB/s eta 0:00:00
Collecting colorlog
  Downloading colorlog-6.7.0-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: packaging>=20.0 in c:\users\zhumh\anaconda3\lib\site-pack
ages (from optuna) (23.1)
Requirement already satisfied: sqlalchemy>=1.3.0 in c:\users\zhumh\anaconda3\lib\site-pa
ckages (from optuna) (1.4.22)
Requirement already satisfied: tqdm in c:\users\zhumh\anaconda3\lib\site-packages (from
optuna) (4.62.3)
Collecting cmaes>=0.10.0
  Downloading cmaes-0.10.0-py3-none-any.whl (29 kB)
Requirement already satisfied: PyYAML in c:\users\zhumh\anaconda3\lib\site-packages (fro
m optuna) (6.0)
Requirement already satisfied: numpy in c:\users\zhumh\anaconda3\lib\site-packages (from
optuna) (1.20.3)
Collecting alembic>=1.5.0
  Downloading alembic-1.12.0-py3-none-any.whl (226 kB)
     ------------------------------------- 226.0/226.0 kB 13.5 MB/s eta 0:00:00
Collecting Mako
  Downloading Mako-1.2.4-py3-none-any.whl (78 kB)
     ---------------------------------------- 78.7/78.7 kB ? eta 0:00:00
Requirement already satisfied: typing-extensions>=4 in c:\users\zhumh\anaconda3\lib\site
-packages (from alembic>=1.5.0->optuna) (4.5.0)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\zhumh\anaconda3\lib\site-pac
kages (from sqlalchemy>=1.3.0->optuna) (1.1.1)
Requirement already satisfied: colorama in c:\users\zhumh\anaconda3\lib\site-packages (f
rom colorlog->optuna) (0.4.4)
Requirement already satisfied: MarkupSafe>=0.9.2 in c:\users\zhumh\anaconda3\lib\site-pa
ckages (from Mako->alembic>=1.5.0->optuna) (1.1.1)
Installing collected packages: Mako, colorlog, cmaes, alembic, optuna
Successfully installed Mako-1.2.4 alembic-1.12.0 cmaes-0.10.0 colorlog-6.7.0 optuna-3.3.
0
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 23.0 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [75]:
```
import optuna
#Define the objective function
def objective(trial):
    # Define range of hyperparameters
    n_estimators = trial.suggest_int('n_estimators', 2, 150)
    max_depth = trial.suggest_int('max_depth', 1, 32, log=True)
    min_samples_split = trial.suggest_float('min_samples_split', 0.1, 1)
    min_samples_leaf = trial.suggest_float('min_samples_leaf', 0.1, 0.5)
    max_features = trial.suggest_categorical('max_features', ['auto', 'sqrt', 'log2'])

    # Initialize and train a RandomForestClassifier with the suggested hyperparameters
    classifier = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        max_features=max_features,
        random_state=90
```

```
    )
    return cross_val_score(classifier, xtrain, ytrain, n_jobs=-1, cv=cv).mean()
```

In [78]:

```
from sklearn.model_selection import cross_val_score
# For regression tasks, you'd use 'minimize'
study = optuna.create_study(direction='maximize')
# For regression tasks, you'd use 'minimize'
study.optimize(objective, n_trials=100)
```

```
[I 2023-10-14 10:43:12,543] A new study created in memory with name: no-name-5041216d-f8
e2-4294-9a6d-bac27437ee6f
[I 2023-10-14 10:43:16,686] Trial 0 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 102, 'max_depth': 27, 'min_samples_split': 0.898321984845962, 'min_
samples_leaf': 0.2818903929854069, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:18,216] Trial 1 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 31, 'max_depth': 1, 'min_samples_split': 0.4141812245368761, 'min_s
amples_leaf': 0.27458814582288515, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:19,530] Trial 2 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 47, 'max_depth': 8, 'min_samples_split': 0.6975042435481382, 'min_s
amples_leaf': 0.483955925087878, 'max_features': 'sqrt'}. Best is trial 0 with value: 0.
6279779754284622.
[I 2023-10-14 10:43:23,942] Trial 3 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 148, 'max_depth': 14, 'min_samples_split': 0.9153236741979063, 'min
_samples_leaf': 0.3054727071223292, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:27,612] Trial 4 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 119, 'max_depth': 6, 'min_samples_split': 0.8427791066074716, 'min_
samples_leaf': 0.41924414425887657, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:31,794] Trial 5 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 138, 'max_depth': 1, 'min_samples_split': 0.5094256545674812, 'min_
samples_leaf': 0.4798343745606777, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:35,524] Trial 6 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 119, 'max_depth': 16, 'min_samples_split': 0.7581943396185549, 'min
_samples_leaf': 0.13726796796768423, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:43:36,877] Trial 7 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 37, 'max_depth': 3, 'min_samples_split': 0.13909856527814415, 'min_
samples_leaf': 0.3846351475143044, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:40,035] Trial 8 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 57, 'max_depth': 27, 'min_samples_split': 0.20766560405754947, 'min
_samples_leaf': 0.28154051994873386, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:43:43,082] Trial 9 finished with value: 0.6279779754284622 and paramete
rs: {'n_estimators': 93, 'max_depth': 1, 'min_samples_split': 0.6514357120587015, 'min_s
amples_leaf': 0.15255790248219148, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:45,894] Trial 10 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 87, 'max_depth': 32, 'min_samples_split': 0.9996986508800568, 'min
_samples_leaf': 0.20635581745197606, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:43:47,088] Trial 11 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 12, 'max_depth': 3, 'min_samples_split': 0.4468362184238444, 'min_
samples_leaf': 0.27663696408862615, 'max_features': 'log2'}. Best is trial 0 with value:
```

```
0.6279779754284622.
[I 2023-10-14 10:43:47,933] Trial 12 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 7, 'max_depth': 2, 'min_samples_split': 0.39514109573786305, 'min_
samples_leaf': 0.22554602776403254, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:50,166] Trial 13 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 67, 'max_depth': 2, 'min_samples_split': 0.6110782039670992, 'min_
samples_leaf': 0.33565143307134826, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:51,356] Trial 14 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 28, 'max_depth': 6, 'min_samples_split': 0.8002142637251646, 'min_
samples_leaf': 0.2336543968800739, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:54,396] Trial 15 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 100, 'max_depth': 1, 'min_samples_split': 0.5617639706734813, 'min
_samples_leaf': 0.3381118643831704, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:43:58,460] Trial 16 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 67, 'max_depth': 10, 'min_samples_split': 0.3649364888150174, 'min
_samples_leaf': 0.18146539483849058, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:01,064] Trial 17 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 79, 'max_depth': 19, 'min_samples_split': 0.7177910173770619, 'min
_samples_leaf': 0.10048731292894925, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:04,916] Trial 18 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 111, 'max_depth': 5, 'min_samples_split': 0.8736585041111146, 'min
_samples_leaf': 0.24932680182736422, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:06,756] Trial 19 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 27, 'max_depth': 10, 'min_samples_split': 0.3097210182507328, 'min
_samples_leaf': 0.2628182571738098, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:09,426] Trial 20 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 49, 'max_depth': 22, 'min_samples_split': 0.5250296038028183, 'min
_samples_leaf': 0.20633929889623717, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:11,273] Trial 21 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 49, 'max_depth': 9, 'min_samples_split': 0.6841543932410237, 'min_
samples_leaf': 0.3163721310258273, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:12,806] Trial 22 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 35, 'max_depth': 16, 'min_samples_split': 0.7546188565615639, 'min
_samples_leaf': 0.35895500157495186, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:14,146] Trial 23 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 18, 'max_depth': 27, 'min_samples_split': 0.6232942456082105, 'min
_samples_leaf': 0.29913413987184145, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:15,931] Trial 24 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 49, 'max_depth': 8, 'min_samples_split': 0.9317618720564579, 'min_
samples_leaf': 0.4979544423920539, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:18,229] Trial 25 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 69, 'max_depth': 12, 'min_samples_split': 0.812014727309154, 'min_
samples_leaf': 0.43641327600123714, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:18,686] Trial 26 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 2, 'max_depth': 20, 'min_samples_split': 0.695248284676889, 'min_s
amples_leaf': 0.25608672310195596, 'max_features': 'sqrt'}. Best is trial 0 with value:
```

```
0.6279779754284622.
[I 2023-10-14 10:44:22,041] Trial 27 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 102, 'max_depth': 7, 'min_samples_split': 0.6018080707051732, 'min
_samples_leaf': 0.3763240710360309, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:24,260] Trial 28 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 41, 'max_depth': 11, 'min_samples_split': 0.4763779351751533, 'min
_samples_leaf': 0.2980794825120822, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:27,044] Trial 29 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 80, 'max_depth': 4, 'min_samples_split': 0.8884904225081436, 'min_
samples_leaf': 0.31154046197909807, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:31,790] Trial 30 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 148, 'max_depth': 14, 'min_samples_split': 0.9408829805251069, 'mi
n_samples_leaf': 0.2741610284450319, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:36,070] Trial 31 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 132, 'max_depth': 13, 'min_samples_split': 0.8361485066656336, 'mi
n_samples_leaf': 0.4242868922835037, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:40,090] Trial 32 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 133, 'max_depth': 7, 'min_samples_split': 0.9920017395541736, 'min
_samples_leaf': 0.44516983641384866, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:44,506] Trial 33 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 149, 'max_depth': 15, 'min_samples_split': 0.7570696011065885, 'mi
n_samples_leaf': 0.46180378192065363, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:48,111] Trial 34 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 121, 'max_depth': 19, 'min_samples_split': 0.8762360065212522, 'mi
n_samples_leaf': 0.3912450412649081, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:52,156] Trial 35 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 121, 'max_depth': 24, 'min_samples_split': 0.7887546266549996, 'mi
n_samples_leaf': 0.40370787016037074, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:54,082] Trial 36 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 57, 'max_depth': 16, 'min_samples_split': 0.830176175288756, 'min_
samples_leaf': 0.36125956057723346, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:44:57,630] Trial 37 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 112, 'max_depth': 31, 'min_samples_split': 0.7268410398252934, 'mi
n_samples_leaf': 0.4706920898846663, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:44:58,675] Trial 38 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 22, 'max_depth': 11, 'min_samples_split': 0.9303674314719561, 'min
_samples_leaf': 0.4095923796044372, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:45:03,444] Trial 39 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 89, 'max_depth': 26, 'min_samples_split': 0.5408080050644859, 'min
_samples_leaf': 0.28719599766932835, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:45:08,006] Trial 40 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 141, 'max_depth': 13, 'min_samples_split': 0.7847830649609338, 'mi
n_samples_leaf': 0.3281213470353705, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:45:12,375] Trial 41 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 131, 'max_depth': 6, 'min_samples_split': 0.8369830234003717, 'min
_samples_leaf': 0.4446933300247564, 'max_features': 'log2'}. Best is trial 0 with value:
```

```
0.6279779754284622.
[I 2023-10-14 10:45:15,620] Trial 42 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 99, 'max_depth': 8, 'min_samples_split': 0.6589809448584121, 'min_
samples_leaf': 0.4824841780821523, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:45:19,271] Trial 43 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 109, 'max_depth': 4, 'min_samples_split': 0.8692141070110962, 'min
_samples_leaf': 0.28710546875170784, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:45:24,511] Trial 44 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 139, 'max_depth': 1, 'min_samples_split': 0.972394170119813, 'min_
samples_leaf': 0.4994846752299192, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:45:29,440] Trial 45 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 126, 'max_depth': 3, 'min_samples_split': 0.9354195008257256, 'min
_samples_leaf': 0.3592197556653419, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:45:31,640] Trial 46 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 59, 'max_depth': 18, 'min_samples_split': 0.9008725903860058, 'min
_samples_leaf': 0.3409521345431082, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:45:33,276] Trial 47 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 41, 'max_depth': 22, 'min_samples_split': 0.7409884467141774, 'min
_samples_leaf': 0.22641423850984033, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:45:37,038] Trial 48 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 94, 'max_depth': 10, 'min_samples_split': 0.8520109200110074, 'min
_samples_leaf': 0.2683613152834057, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:45:40,690] Trial 49 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 84, 'max_depth': 32, 'min_samples_split': 0.7861085885629774, 'min
_samples_leaf': 0.23868607213179005, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:45:45,336] Trial 50 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 116, 'max_depth': 5, 'min_samples_split': 0.9014652869845906, 'min
_samples_leaf': 0.31119490626415947, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:45:50,320] Trial 51 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 139, 'max_depth': 2, 'min_samples_split': 0.46775882469634056, 'mi
n_samples_leaf': 0.4657570149416542, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:45:54,972] Trial 52 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 126, 'max_depth': 1, 'min_samples_split': 0.5779050435308845, 'min
_samples_leaf': 0.4223957190773576, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:46:00,610] Trial 53 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 144, 'max_depth': 9, 'min_samples_split': 0.37180007574613416, 'mi
n_samples_leaf': 0.4827135611655932, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:01,500] Trial 54 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 14, 'max_depth': 16, 'min_samples_split': 0.5216551607192437, 'min
_samples_leaf': 0.45271254116125953, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:02,823] Trial 55 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 34, 'max_depth': 2, 'min_samples_split': 0.641014510269146, 'min_s
amples_leaf': 0.2502552256971975, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:46:07,104] Trial 56 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 106, 'max_depth': 22, 'min_samples_split': 0.6868127382020212, 'mi
n_samples_leaf': 0.20837494098746556, 'max_features': 'auto'}. Best is trial 0 with valu
```

```
e: 0.6279779754284622.
[I 2023-10-14 10:46:09,904] Trial 57 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 71, 'max_depth': 11, 'min_samples_split': 0.4409297214478312, 'min
_samples_leaf': 0.436706860569629, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:46:11,344] Trial 58 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 26, 'max_depth': 14, 'min_samples_split': 0.5761505536846592, 'min
_samples_leaf': 0.4862404902367811, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:46:16,933] Trial 59 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 136, 'max_depth': 18, 'min_samples_split': 0.8076161199655025, 'mi
n_samples_leaf': 0.26860800942389806, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:22,405] Trial 60 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 145, 'max_depth': 12, 'min_samples_split': 0.281891787875714, 'min
_samples_leaf': 0.4607262911068308 3, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:26,790] Trial 61 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 116, 'max_depth': 25, 'min_samples_split': 0.7093630802721685, 'mi
n_samples_leaf': 0.14383175644076235, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:31,537] Trial 62 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 125, 'max_depth': 9, 'min_samples_split': 0.8563989428079881, 'min
_samples_leaf': 0.1779716931715483, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:46:37,325] Trial 63 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 150, 'max_depth': 17, 'min_samples_split': 0.7747856757789537, 'mi
n_samples_leaf': 0.24182928123496153, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:41,560] Trial 64 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 117, 'max_depth': 21, 'min_samples_split': 0.8190330867687513, 'mi
n_samples_leaf': 0.3830364091830908, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:45,484] Trial 65 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 103, 'max_depth': 14, 'min_samples_split': 0.7596843730934127, 'mi
n_samples_leaf': 0.2601711859555983, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:46:50,410] Trial 66 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 129, 'max_depth': 7, 'min_samples_split': 0.8983427479483347, 'min
_samples_leaf': 0.3052884890890965, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:46:52,173] Trial 67 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 43, 'max_depth': 28, 'min_samples_split': 0.734009981261557, 'min_
samples_leaf': 0.28151189007143257, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:46:57,310] Trial 68 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 135, 'max_depth': 19, 'min_samples_split': 0.6657679389760729, 'mi
n_samples_leaf': 0.3243182788912993, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:01,135] Trial 69 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 96, 'max_depth': 15, 'min_samples_split': 0.8093601368635577, 'min
_samples_leaf': 0.34625636753669164, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:05,461] Trial 70 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 63, 'max_depth': 24, 'min_samples_split': 0.6251571085417607, 'min
_samples_leaf': 0.29179770333056104, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:06,923] Trial 71 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 35, 'max_depth': 6, 'min_samples_split': 0.1019268356747601 9, 'min
_samples_leaf': 0.3723567638999897, 'max_features': 'sqrt'}. Best is trial 0 with value:
```

```
0.6279779754284622.
[I 2023-10-14 10:47:09,198] Trial 72 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 51, 'max_depth': 1, 'min_samples_split': 0.4310713904599617З, 'min
_samples_leaf': 0.31811315027481607, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:10,622] Trial 73 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 33, 'max_depth': 5, 'min_samples_split': 0.3297426367004505, 'min_
samples_leaf': 0.3939106707486854, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:11,721] Trial 74 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 12, 'max_depth': 8, 'min_samples_split': 0.5018132618610055, 'min_
samples_leaf': 0.2993953012008024, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:13,740] Trial 75 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 46, 'max_depth': 12, 'min_samples_split': 0.4176039263607171, 'min
_samples_leaf': 0.42035253462262046, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:15,230] Trial 76 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 29, 'max_depth': 3, 'min_samples_split': 0.6090739594958068, 'min_
samples_leaf': 0.34941724265226803, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:17,586] Trial 77 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 54, 'max_depth': 29, 'min_samples_split': 0.2534057783912965, 'min
_samples_leaf': 0.3306089979030816, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:22,371] Trial 78 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 121, 'max_depth': 21, 'min_samples_split': 0.7019993103140589, 'mi
n_samples_leaf': 0.27221003446150216, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:25,801] Trial 79 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 87, 'max_depth': 17, 'min_samples_split': 0.5476381454844005, 'min
_samples_leaf': 0.4293036481385116, 'max_features': 'log2'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:28,870] Trial 80 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 77, 'max_depth': 19, 'min_samples_split': 0.4854497288949161, 'min
_samples_leaf': 0.4526869406986166, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:30,520] Trial 81 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 38, 'max_depth': 28, 'min_samples_split': 0.8470510944146493, 'min
_samples_leaf': 0.282954054547299, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:36,367] Trial 82 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 144, 'max_depth': 25, 'min_samples_split': 0.8821825494645882, 'mi
n_samples_leaf': 0.41078784649129857, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:37,496] Trial 83 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 21, 'max_depth': 23, 'min_samples_split': 0.960429360500898, 'min_
samples_leaf': 0.2587198556942357, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:40,403] Trial 84 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 47, 'max_depth': 32, 'min_samples_split': 0.16376760683581626, 'mi
n_samples_leaf': 0.29415707099294175, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:42,720] Trial 85 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 57, 'max_depth': 20, 'min_samples_split': 0.9222627077999057, 'min
_samples_leaf': 0.30692215344653034, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:47,422] Trial 86 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 111, 'max_depth': 10, 'min_samples_split': 0.3980544836169786, 'mi
n_samples_leaf': 0.3177134711365873, 'max_features': 'sqrt'}. Best is trial 0 with valu
```

```
e: 0.6279779754284622.
[I 2023-10-14 10:47:50,054] Trial 87 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 62, 'max_depth': 28, 'min_samples_split': 0.8221424593937806, 'min
_samples_leaf': 0.2752822692496853, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:47:51,769] Trial 88 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 39, 'max_depth': 13, 'min_samples_split': 0.4594034427047169, 'min
_samples_leaf': 0.33617334795748255, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:54,963] Trial 89 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 74, 'max_depth': 23, 'min_samples_split': 0.8634968023507628, 'min
_samples_leaf': 0.47387078410991096, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:56,317] Trial 90 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 28, 'max_depth': 26, 'min_samples_split': 0.5151352680851901, 'min
_samples_leaf': 0.49076753446384924, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:47:59,635] Trial 91 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 84, 'max_depth': 1, 'min_samples_split': 0.6482314562734385, 'min_
samples_leaf': 0.4748975749344348, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:48:03,870] Trial 92 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 106, 'max_depth': 7, 'min_samples_split': 0.6721426524647828, 'min
_samples_leaf': 0.2497558755418164, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:48:10,283] Trial 93 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 90, 'max_depth': 16, 'min_samples_split': 0.5894534641170066, 'min
_samples_leaf': 0.13188370413945155, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:48:13,950] Trial 94 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 100, 'max_depth': 4, 'min_samples_split': 0.7296961259739357, 'min
_samples_leaf': 0.3724468339628183, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:48:17,763] Trial 95 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 95, 'max_depth': 9, 'min_samples_split': 0.683399920254363, 'min_s
amples_leaf': 0.2843391071192418, 'max_features': 'sqrt'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:48:26,217] Trial 96 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 129, 'max_depth': 6, 'min_samples_split': 0.5383103282423337, 'min
_samples_leaf': 0.22664410655469286, 'max_features': 'log2'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:48:31,466] Trial 97 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 138, 'max_depth': 1, 'min_samples_split': 0.712243193401329, 'min_
samples_leaf': 0.49843491187513234, 'max_features': 'auto'}. Best is trial 0 with value:
0.6279779754284622.
[I 2023-10-14 10:48:36,831] Trial 98 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 147, 'max_depth': 30, 'min_samples_split': 0.754022552682373, 'min
_samples_leaf': 0.29469723889907634, 'max_features': 'sqrt'}. Best is trial 0 with valu
e: 0.6279779754284622.
[I 2023-10-14 10:48:45,005] Trial 99 finished with value: 0.6279779754284622 and paramet
ers: {'n_estimators': 123, 'max_depth': 20, 'min_samples_split': 0.5595920451910639, 'mi
n_samples_leaf': 0.2656506419519053, 'max_features': 'auto'}. Best is trial 0 with valu
e: 0.6279779754284622.
```

In [79]:

```
#Printing the best hyperparameters and their corresponding cross-validation score
best_params = study.best_params
best_score = study.best_value
```

```
print(f"Best parameters: {best_params}")
print(f"Best cross-validation score: {best_score}")
```

```
Best parameters: {'n_estimators': 102, 'max_depth': 27, 'min_samples_split': 0.898321984
845962, 'min_samples_leaf': 0.2818903929854069, 'max_features': 'sqrt'}
Best cross-validation score: 0.6279779754284622
```

In [84]:

```
#Use the best hyperparameters you obtained from the Optuna study to create a RandomFore
best_rfc = RandomForestClassifier(**best_params, random_state=90)
best_rfc.fit(xtrain, ytrain)

y_score_best = best_rfc.predict_proba(xtest)[:, 1]
y_pred_best = best_rfc.predict(xtest)
```

2XGBoost

In [87]:

```
pip install xgboost
```

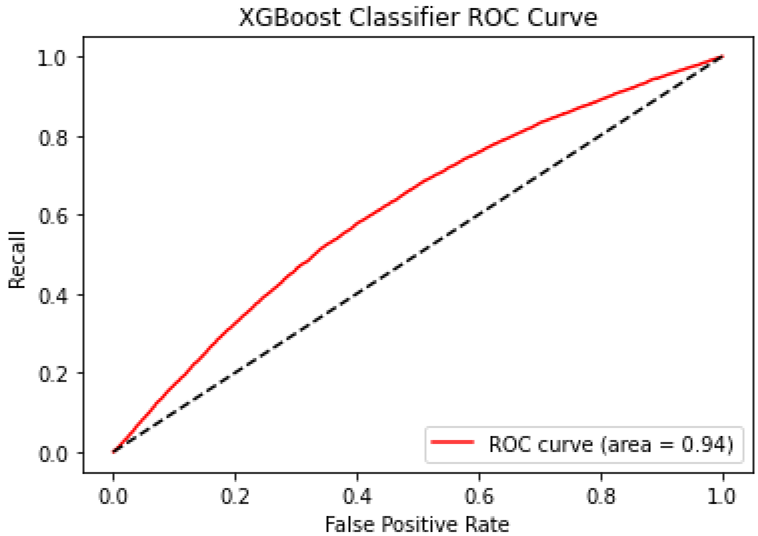
```
Note: you may need to restart the kernel to use updated packages.Collecting xgboost
  Downloading xgboost-2.0.0-py3-none-win_amd64.whl (99.7 MB)
     --------------------------------------- 99.7/99.7 MB 21.1 MB/s eta 0:00:00
Requirement already satisfied: scipy in c:\users\zhumh\anaconda3\lib\site-packages (from
xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\zhumh\anaconda3\lib\site-packages (from
xgboost) (1.20.3)
Installing collected packages: xgboost
Successfully installed xgboost-2.0.0

[notice] A new release of pip is available: 23.0 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [88]:

```
from xgboost import XGBClassifier
#Initialize the XGBoostClassifier with 100 estimators and a random seed
xgbr=XGBClassifier(n_estimators=100,random_state=90)
#Calculate the mean cross-validation score for the XGBoostClassifier
xgbr_score=cvs(xgbr,xtrain,ytrain,cv=cv).mean()
xgbr.fit(xtrain,ytrain)
#Generate predicted probabilities for the positive class (class 1) for the test data
y_score=xgbr.predict_proba(xtest)[:,1]
xgbr_pred=xgbr.predict(xtest)
FPR, recall, thresholds = roc_curve(ytest,y_score, pos_label=1)
xgbr_auc = AUC(ytest,y_score)
```

In [94]:

```
#Plotting the ROC curve for the XGBoost classifierplt.figure(figsize=(8,8))
plt.plot(FPR, recall, color='red',label='ROC curve (area = %0.2f)' % xgbr_auc)
plt.plot([0, 1], [0, 1], color='black', linestyle='--')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('Recall')
plt.title('XGBoost Classifier ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



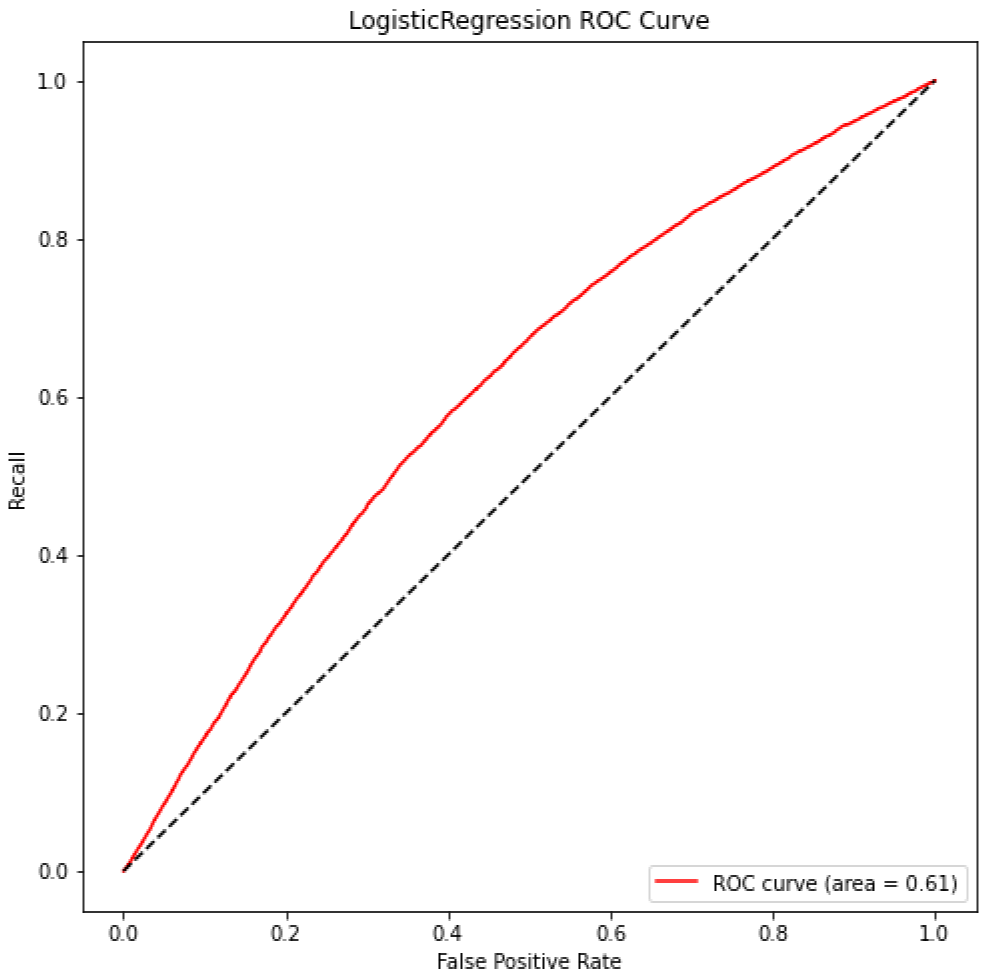
Logistic Regression

In [90]:

```
from sklearn.linear_model import LogisticRegression as LR
#Initialize the Logistic Regression classifier with specific settings (L2 penalty, libl
lr = LR(penalty='l2',solver='liblinear',max_iter=1000)
#Calculate the mean cross-validation score for the Logistic Regression classifier
lr_score=cvs(lr,xtrain,ytrain,cv=cv).mean()
lr.fit(xtrain,ytrain)
#Generate predicted probabilities for the positive class (class 1) for the test data
y_score=lr.predict_proba(xtest)[:,1]
lr_pred=lr.predict(xtest)
FPR, recall, thresholds = roc_curve(ytest,y_score, pos_label=1)
lr_auc = AUC(ytest,y_score)
```

In [91]:

```
#Plotting the ROC curve for the Logistic Regression classifier
plt.figure(figsize=(8,8))
plt.plot(FPR, recall, color='red',label='ROC curve (area = %0.2f)' % lr_auc)
plt.plot([0, 1], [0, 1], color='black', linestyle='--')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('Recall')
plt.title('LogisticRegression ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



In [92]:

```
# Print a summary of the classification performance of each of the three classifiers on
from sklearn.metrics import classification_report as CR
print('Random Forest'.center(50), CR(ytest,rfc_pred),sep='\n')
print('XGBoost'.center(55),CR(ytest,xgbr_pred),sep='\n')
print('Logistic Regression'.center(50),CR(ytest,lr_pred),sep='\n')
```

```
                  Random Forest
              precision    recall  f1-score   support

           0       0.88      0.93      0.90     22684
           1       0.87      0.78      0.82     13133

    accuracy                           0.88     35817
   macro avg       0.87      0.85      0.86     35817
weighted avg       0.88      0.88      0.87     35817

                        XGBoost
              precision    recall  f1-score   support

           0       0.88      0.91      0.90     22684
           1       0.84      0.79      0.82     13133

    accuracy                           0.87     35817
   macro avg       0.86      0.85      0.86     35817
weighted avg       0.87      0.87      0.87     35817

               Logistic Regression
```

```
              precision    recall  f1-score   support

           0       0.66      0.85      0.74     22684
           1       0.49      0.25      0.34     13133

    accuracy                           0.63     35817
   macro avg       0.58      0.55      0.54     35817
weighted avg       0.60      0.63      0.59     35817
```

In [93]:

```
#Displays the model scores and AUC values for each of the three models
score={'Model_score':[rfc_score,xgbr_score,lr_score],'Auc_area':[rfc_auc,xgbr_auc,lr_au
score_com=pd.DataFrame(data=score,index=['RandomForest','XGBoost','LogisticRegression']
score_com.sort_values(by=['Model_score'],ascending=False)
```

Out[93]:

	Model_score	Auc_area
RandomForest	0.877078	0.946692
XGBoost	0.870628	0.942924
LogisticRegression	0.639477	0.613851